Topics Covered:

- Manipulator Analysis
- Forward Kinematics
- Inverse Kinematics
- Manipulator Jacobian
- Singularity Analysis
- Trajectory Design
- Wrenches and Forces
- Dynamics and Control
- Equations Page

Additional Reading:

- MLS Chapter 2-3
- LP Chapter 2-9

Note: You are also responsible for the material covered in the Midterm Review (Lecture 10). These lecture notes only cover the material from Lecture 11 to Lecture 24. NOTE: This review is not necessarily exhaustive, but it covers the main topics that you should focus on for the final exam.

Note 2: The best practice problems to study from are your homeworks (including code!) and the example problems from lecture. A few small practice problems are also available here (will be the in-class exercise on December 3rd): Kahoot Review

Equations to Memorize

The following are equations that you can be expected to memorize: The twist for a revolute joint:

$$\xi = \begin{bmatrix} -\hat{\omega} \times q \\ \hat{\omega} \end{bmatrix}$$

The twist for a prismatic joint:

$$\xi = \begin{bmatrix} \hat{v} \\ 0 \end{bmatrix}$$

The form of the twist hat:

$$\hat{\xi} = \begin{bmatrix} [\omega]_\times & v \\ 0 & 0 \end{bmatrix}$$

The product of exponentials forward-kinematics method formula:

$$g_e(\vec{\theta}) = e^{\hat{\xi}_1\theta_1} e^{\hat{\xi}_2\theta_2} \cdots e^{\hat{\xi}_m\theta_m} g_0$$

The product of HTM forward-kinematics method formula:

$$g_e(\vec{\theta}) = g_1(\theta_1) g_2(\theta_2) \cdots g_m(\theta_m) g_{m+1}$$

The adjoint operation:

$$\text{Ad}_{g_1} g_2 = g_1 g_2 g_1^{-1}$$

The inverse adjoint operation:

$$\text{Ad}_{g_1}^{-1} g_2 = g_1^{-1} g_2 g_1$$

Manipulator Analysis

We typically operate in one of two spaces: Joint Space ($\vec{\theta} \in M$) or Configuration Space ($g \in SE(n)$).

Joint space can be mathematically represented as a higher dimensional Torus of r revolute joints combined with the Cartesian space of p prismatic/helical joints:

$$M = T^r \times \mathbb{R}^p$$
$$r : \text{\# revolute joints}$$
$$p : \text{\# prismatic/helical joints}$$

Configuration space is mathematically represented through three definitions of workspace.

Normal workspace (complete workspace) is defined as:

$$W = \{g_e(\vec{\theta}) \in SE(n) \mid \vec{\theta} \in M)\}$$

Reachable workspace is defined as:

$$W_r = \{p_e(\vec{\theta}) \in E(n) \mid \vec{\theta} \in M\}$$

And lastly, dextrous workspace is defined as:

$$W_d = \{p_e(\vec{\theta}) \in E(n) \mid \ \forall R \in SO(n),\ \exists \vec{\theta} \in M \ s.t.\ g_e(\vec{\theta} = (p_e, R)\}$$

Forward Kinematics

There are two main methods we leveraged for forward kinematics: the product of homogeneous transformation matrices (or sometimes called the product of Lie groups) and the product of exponentials.

The product of homogeneous transformation matrices method uses the formula (for a manipulator with m joints):

$$g_e(\vec{\theta}) = g_1(\theta_1)g_2(\theta_2)\cdots g_m(\theta_m)g_{m+1}$$

The product of exponentials method uses the formula (for a manipulator with m joints):

$$g_e(\vec{\theta}) = e^{\hat{\xi}_1\theta_1}e^{\hat{\xi}_2\theta_2}\cdots e^{\hat{\xi}_m\theta_m}g_0$$

For the product of exponential method, the twists are obtained using the formulas:

$$\xi_i = \begin{bmatrix} v_i \\ \omega_i \end{bmatrix} = \begin{cases} \begin{bmatrix} -\omega_i \times q_i \\ \omega_i \end{bmatrix} & \text{if revolute} \\ \\ \begin{bmatrix} v_i \\ 0 \end{bmatrix} & \text{if prismatic} \end{cases}$$

In the above formulas for a revolute joint, ω_i is the unit axis of rotation and q_i is any point on the axis of rotation. For the formula for the prismatic joint, v_i is the unit axis of translation. All variables are relative to the spatial (fixed) frame, and are taken at the zero configuration.

The twist is "hatted" by converting the vector into homogeneous form:

$$\xi_i = \begin{bmatrix} v_i \\ \omega_i \end{bmatrix} \quad \underset{\text{hatting}}{\rightarrow} \quad \hat{\xi}_i = \begin{bmatrix} [\omega_i]_\times & v_i \\ 0 & 0 \end{bmatrix}$$

This homogeneous form can be put back into vector form by "unhatting" or "veeing" the matrix:

$$\hat{\xi}_i = \begin{bmatrix} [\omega_i]_\times & v_i \\ 0 & 0 \end{bmatrix} \quad \underset{\text{unhatting}}{\rightarrow} \quad \xi_i = (\hat{\xi}_i)^\vee = \begin{bmatrix} v_i \\ \omega_i \end{bmatrix}$$

Inverse Kinematics

There are three main methods that we covered for inverse kinematics: the geometric approach, Newton-Raphson (an iterative numerical approach), and an optimization-based approach. We also covered resolved-rate trajectory generation which can be thought of as an approach to "inverse kinematics" (by taking a linear integration step of our desired velocity), but we will cover this later in the trajectory design section.

Geometric Approach

For simple manipulator arms, we can use the geometric approach to directly solve for joint angles required to achieve a desired end-effector position.

This is accomplished by drawing the triangle formed by the end-effector position and a two-link manipulator arm, and using the law of cosines to solve for the corresponding joint angles. We can then use our standard equations for either the "Lefty" (also known as "Elbow-Up") configuration:

$$\theta_1 = \gamma + \alpha$$
$$\theta_2 = \beta - \pi$$

For the "Righty" (also known as "Elbow-Down") configuration we instead use the equations:

$$\theta_1 = \gamma - \alpha$$
$$\theta_2 = \pi - \beta$$

With α and β solved for using the law of cosines:

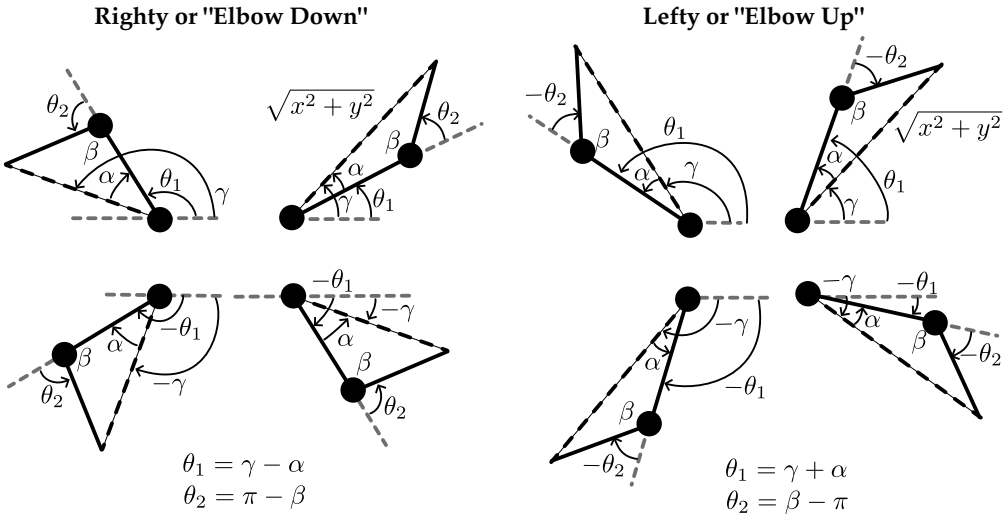
$$\alpha = \cos^{-1}\left(\frac{x^2 + y^2 + l_1^2 - l_2^2}{2l_1\sqrt{x^2 + y^2}}\right)$$
$$\beta = \cos^{-1}\left(\frac{l_1^2 + l_2^2 - x^2 - y^2}{2l_1 l_2}\right)$$

And with γ solved for using the two-argument arctangent function:

$$\gamma = \text{atan2}(y, x)$$

We can ignore $\pm\cos^{-1}$ as long as you pay careful attention to the arguments of arccosine and which solution you utilize in which quadrant.

For clarity, there's each of the labels for the Lefty and Righty configurations when the manipulator is in each of the four quadrants:



Newton-Raphson

The Newton-Raphson method is an iterative root-finding algorithm that can be used for inverse kinematics. The general equation for Newton-Raphson is:

$$\theta_{k+1} = \theta_k + \frac{\partial f(\theta_k)}{\partial \theta}(\theta_k)(x_d - f(\theta_0))$$
$$\theta_{k+1} = \theta_k + J^{\dagger}(\theta_k)(x_d - f(\theta_0)),$$

with $f(\theta)$ being the forward kinematics mapping of the end-effector position, and $J^{\dagger}$ being the pseudoinverse of the manipulator Jacobian (discussed in the next section). This algorithm is continued until the difference between θ_{k+1} and θ_k is less than a specified tolerance.

Optimization-Based Inverse Kinematics

Lastly, we commonly leverage modern optimization techniques to solve for inverse kinematics. This is done by minimizing the error between the desired end-effector position and the actual end-effector position by solving an optimization problem of the form:

$$\theta^* = \underset{\theta}{\text{argmin}} \ \|x_d - f(\theta)\|^2$$
$$\text{subject to } \theta_{\text{min}} \leq \theta \leq \theta_{\text{max}}$$

These optimization problems are commonly solved by numerically approximating the gradient of the cost function and using a gradient-based optimization algorithm such as gradient descent or L-BFGS.

Manipulator Jacobian

The manipulator Jacobian is a matrix that maps joint velocities to end-effector velocities:

$$\xi_s = J_s(\theta)\dot{\theta} \qquad \text{(Spatial Manipulator Jacobian)}$$

or

$$\xi_b = J_b(\theta)\dot{\theta} \qquad \text{(Body Manipulator Jacobian)}$$

This manipulator Jacobian is a special case of the more general Jacobian matrix which captures the linear sensitivity of a function (i.e., $f(x) = [f_1(x), \cdots, f_n(x)]^\top$) to changes in its input (i.e., $x = [x_1, \cdots, x_m]^\top$) in terms of the velocities:

$$\frac{df(x)}{dt} = J(x)\frac{dx}{dt}$$

with the general Jacobian (derived using the chain rule) defined as:

$$J(x) = \frac{\partial f(x)}{\partial x} = Df(x) = \begin{bmatrix} \frac{\partial f_1(x)}{\partial x_1} & \frac{\partial f_1(x)}{\partial x_2} & \cdots & \frac{\partial f_1(x)}{\partial x_m} \\ \frac{\partial f_2(x)}{\partial x_1} & \frac{\partial f_2(x)}{\partial x_2} & \cdots & \frac{\partial f_2(x)}{\partial x_m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n(x)}{\partial x_1} & \frac{\partial f_n(x)}{\partial x_2} & \cdots & \frac{\partial f_n(x)}{\partial x_m} \end{bmatrix} \in \mathbb{R}^{n \times m}$$

For the manipulator Jacobian, there are several formulas associated with it, but it's most general definition (that is used to derive the other formulas) is:

$$\begin{aligned} \xi_s &= J_s(\theta)\dot{\theta} \\ \dot{g}_e g_e^{-1} &= J_s(\theta)\dot{\theta} \\ \frac{\partial g_e}{\partial \theta}\frac{\partial \theta}{\partial t} g_e^{-1} &= J_s(\theta)\dot{\theta} \\ \left[(\tfrac{\partial g_e}{\partial \theta_1} g_e^{-1})^\vee \quad \cdots \quad (\tfrac{\partial g_e}{\partial \theta_m} g_e^{-1})^\vee \right] \dot{\theta} &= J_s(\theta)\dot{\theta} \\ &\Rightarrow \\ J_s(\theta) &= \left[(\tfrac{\partial g_e}{\partial \theta_1} g_e^{-1})^\vee, \cdots, (\tfrac{\partial g_e}{\partial \theta_m} g_e^{-1})^\vee \right] \end{aligned}$$

and

$$\begin{aligned} \xi_b &= J_b(\theta)\dot{\theta} \\ g_e^{-1}\dot{g}_e &= J_b(\theta)\dot{\theta} \\ g_e^{-1}\frac{\partial g_e}{\partial \theta}\frac{\partial \theta}{\partial t} &= J_b(\theta)\dot{\theta} \\ \left[(g_e^{-1}\tfrac{\partial g_e}{\partial \theta_1})^\vee \quad \cdots \quad (g_e^{-1}\tfrac{\partial g_e}{\partial \theta_m})^\vee \right] \dot{\theta} &= J_b(\theta)\dot{\theta} \\ &\Rightarrow \\ J_b(\theta) &= \left[(g_e^{-1}\tfrac{\partial g_e}{\partial \theta_1})^\vee \quad \cdots \quad (g_e^{-1}\tfrac{\partial g_e}{\partial \theta_m})^\vee \right] \end{aligned}$$

We showed in class how these formulas can be manipulated (mainly by plugging in our product of Lie groups or product of exponentials formulas for g_e) to derive the other formulas that lend themselves better for practical computation:

$$\begin{aligned} J_s(\theta) &= \begin{bmatrix} \xi_1 & \mathrm{Ad}_{e^{\hat{\xi}_1\theta_1}}\xi_2 & \cdots & \mathrm{Ad}_{e^{\hat{\xi}_1\theta_1}\cdots e^{\hat{\xi}_{m-1}\theta_{m-1}}}\xi_m \end{bmatrix} \\ &= \begin{bmatrix} \frac{\partial g_1}{\partial\theta_1}g_1^{-1} & \mathrm{Ad}_{g_1}\frac{\partial g_2}{\partial\theta_2}g_2^{-1} & \cdots & \mathrm{Ad}_{g_1 g_2\cdots g_{m-1}}\frac{\partial g_m}{\partial\theta_m}g_m^{-1} \end{bmatrix} \end{aligned}$$

and

$$\begin{aligned} J_b(\theta) &= \begin{bmatrix} \mathrm{Ad}^{-1}_{e^{\hat{\xi}_1\theta_1}\cdots e^{\hat{\xi}_m\theta_m}g_0}\xi_1 & \cdots & \mathrm{Ad}^{-1}_{e^{\hat{\xi}_m\theta_m}g_0}\xi_m \end{bmatrix} \\ &= \begin{bmatrix} \mathrm{Ad}^{-1}_{g_2\cdots g_{m+1}}g_1^{-1}\frac{\partial g_1}{\partial\theta_1} & \cdots & \mathrm{Ad}^{-1}_{g_{m+1}}g_m^{-1}\frac{\partial g_m}{\partial\theta_m} \end{bmatrix} \end{aligned}$$

There are two ways to compute these adjoints. The first is using the "matrix Adjoint" formulas:

$$\mathrm{Ad}_g = \begin{bmatrix} R & [d]_\times R \\ 0 & R \end{bmatrix} \quad \text{assuming } \xi := \begin{bmatrix} v \\ \omega \end{bmatrix} \qquad \text{(Our assumed convention for } \xi\text{)}$$

and

$$\mathrm{Ad}_g^{-1} = \begin{bmatrix} R^T & -R^T[d]_\times \\ 0 & R^T \end{bmatrix}$$

So for example, if we wanted to use the matrix Adjoint to compute $\xi_2' = \mathrm{Ad}_{\exp(\hat{\xi}_1\theta_1)}\xi_2$, we would first assign the input to the adjoint operation as some matrix:

$$e1 = \exp(\hat{\xi}_1\theta_1) = \begin{bmatrix} R_1 & d_1 \\ 0 & 1 \end{bmatrix}$$

and then do the following matrix multiplication:

$$\begin{aligned} \xi_2' &= \mathrm{Ad}_{e1}\xi_2 \\ &= \begin{bmatrix} R_1 & [d_1]_\times R_1 \\ 0 & R_1 \end{bmatrix}\xi_2 \end{aligned}$$

However, I recommend using the official definition of an adjoint since it's more general (despite being more computationally complex):

$$\begin{aligned} \xi_2' &= \mathrm{Ad}_{\exp(\hat{\xi}_1\theta_1)}\xi_2 \\ &= \left(\exp(\hat{\xi}_1\theta_1)\hat{\xi}_2\left(\exp(\hat{\xi}_1\theta_1)\right)^{-1}\right)^\vee \end{aligned}$$

Another example of how to do this adjoint operation for the third spatial twist column would be:

$$\begin{aligned} \xi_3' &= \mathrm{Ad}_{\exp(\hat{\xi}_1\theta_1)\exp(\hat{\xi}_2\theta_2)}\xi_3 \\ &= \left(\left(\exp(\hat{\xi}_1\theta_1)\exp(\hat{\xi}_2\theta_2)\right)\hat{\xi}_3\left(\exp(\hat{\xi}_1\theta_1)\exp(\hat{\xi}_2\theta_2)\right)^{-1}\right)^\vee \end{aligned}$$

Lastly, for the body Jacobian, our inverse matrix adjoint is:

$$\begin{aligned}\xi_i^\dagger &= \text{Ad}^{-1}_{\underbrace{\exp(\hat{\xi}_i\theta_i)\cdots\exp(\hat{\xi}_n\theta_n)g_0}_{g}}\xi_i \\ &= \text{Ad}_g^{-1}\xi_i \\ &= \begin{bmatrix} R^T & -R^T[d]_\times \\ 0 & R^T \end{bmatrix}\xi_i\end{aligned}$$

or the general inverse adjoint operation is:

$$\xi_i^\dagger = \left(\left(\exp(\hat{\xi}_i\theta_i)\cdots\exp(\hat{\xi}_n\theta_n)g_0\right)^{-1}\hat{\xi}_i\left(\exp(\hat{\xi}_i\theta_i)\cdots\exp(\hat{\xi}_n\theta_n)g_0\right)\right)^\vee$$

Inverting the Jacobian

There are many uses of the manipulator Jacobian, but one ofthe main ways that we use it is to solve for the joint velocities required to achieve a desired end-effector velocity. This is done by inverting the manipulator Jacobian to obtain:

$$\dot{\theta} = J^{-1}(\theta)\xi$$

When the Jacobian matrix is not square, we must take it's pseudo-inverse rather than the inverse to obtain:

$$\dot{\theta} = J^\dagger(\theta)\xi$$

This pseudo-inverse can be computed using Moore-Penrose pseudo-inverse (assuming $J \in \mathbb{R}^{n\times m}$):

$$J^\dagger = \begin{cases} J^\top(JJ^\top)^{-1} & \text{if } J \text{ is "fat"}(n < m) \\ (J^\top J)^{-1}J^\top & \text{if } J \text{ is "tall"}(m < n) \end{cases}$$

Which inverse/pseudo-inverse we use relates to the functional class of our manipulator:

1. Kinematically Sufficient ($n = m$): Equal dimension for task space and joint space
2. Kinematically Insufficient ($n > m$): Higher dimension task space than joint space
3. Kinematically Redundant ($n < m$): Higher dimension joint space than task space

Note that we should be able to avoid the kinematically insufficient case by reducing the dimensionality of the task space to match the joint space.

Singularity Analysis

A manipulator is at a singular configuration when the end-effector instantaneously loses a degree of freedom in it's mobility. We can solve for our singular configurations by finding all $\theta \in M$ such that the associated manipulator Jacobian matrices lose rank. Note that the rank of a matrix is the number of linearly independent columns. We also know that a square matrix loses rank when its determinant is zero. Thus, we can solve for singular configurations by finding all $\theta \in M$ such that:

$$\det(J) = 0 \qquad \text{(for kinematically-sufficient arms)}$$
$$\det(JJ^\top) = 0 \qquad \text{(for kinematically-redundant arms)}$$
$$\det(J^\top J) = 0 \qquad \text{(for kinematically-insufficient arms)}$$

Trajectory Design

We can design either joint-space or task-space trajectories for our manipulator using polynomials. The degree of our polynomial will dictate how smooth our resulting motion is. We typically use cubic polynomials (degree 3) because they are the lowest order polynomial that will result in smooth position, velocity profiles, and acceleration profiles (no instantaneous jumps). However, these polynomials will not guarantee smooth jerk profiles. If you want smooth jerk profiles, we can scale our polynomials up to start and end with zero acceleration, and define our polynomials as quartic (degree 4) polynomials.

But we typically use a cubic polynomial, defined as:

$$p(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3$$

As long as we have four constraints to impose on our desired trajectory, we can solve for the coefficients of our polynomial. If we take the constraints to be the initial and final position (p_i, and p_f), and the initial and final velocity ($\dot{p}_i$, and $\dot{p}_f$), the coefficients are:

$$\begin{aligned}
a_0 &= p_i \\
a_1 &= \dot{p}_i \\
a_2 &= \frac{3(p_f - p_i)}{T^2} - \frac{2\dot{p}_i}{T} - \frac{\dot{p}_f}{T} \\
a_3 &= \frac{-2(p_f - p_i)}{T^3} + \frac{(\dot{p}_i + \dot{p}_f)}{T^2}
\end{aligned}$$

with T being the total duration of the trajectory, assumed to start at $t_0 = 0$ and end at $t_f = T$.

Further, if we assume that we start and end this trajectory at rest (i.e., $\dot{p}_i = \dot{p}_f = 0$), we can

simplify the above equations to:

$$\begin{aligned} a_0 &= p_i \\ a_1 &= 0 \\ a_2 &= 3\frac{(p_f - p_i)}{T^2} \\ a_3 &= -2\frac{(p_f - p_i)}{T^3} \end{aligned}$$

These polynomials can be used to generate joint-space trajectories by applying them to the joint angles, or task-space trajectories by applying them to the end-effector position.

Joint-Space Trajectories

For joint-space planning, we construct each joint trajectory independently as a polynomial for joint positions, and take the polynomial derivative for joint velocities. The procedure is as follows:

1. Assume we are given a starting and ending coordinates $\vec{\theta}_i^*$ and $\vec{\theta}_f^*$, and a total time duration T.
2. The desired position for each joint j is:

   $$\theta_j(t) = a_{0j} + a_{1j}t + a_{2j}t^2 + a_{3j}t^3$$

   with the coefficients:

   $$\begin{aligned} a_{0j} &= \theta_{ij}^* \\ a_{1j} &= 0 \\ a_{2j} &= 3\frac{(\theta_{fj}^* - \theta_{ij}^*)}{T^2} \\ a_{3j} &= -2\frac{(\theta_{fj}^* - \theta_{ij}^*)}{T^3} \end{aligned}$$

   Note that this trajectory is only valid for times within the domain $[0, T]$.
3. The desired velocity for each joint j is:

   $$\dot{\theta}_j(t) = a_{1j} + 2a_{2j}t + 3a_{3j}t^2$$

Task-Space Trajectories

If instead we want to generate a task-space trajectory, we apply the polynomial to the end-effector pose and implement inverse-kinematics to translate this desired pose to desired joint angles. The procedure is as follows:

1. Assume we are given a desired trajectory for our end-effector pose $g_e^*(t)$ given as a series of waypoints $g_e^*(t_k)$ at times $t_k \in [0, t_f]$.

2. If we also are given an associated velocity trajectory ($\dot{g}_e^*(t)$), we can obtain the desired twist trajectory $\xi^(t_k)$ as either:

$$\xi^b(t_k) = g_e^*(t_k)^{-1}\dot{g}_e^*(t_k)$$
$$\text{or}$$
$$\xi^s(t_k) = \dot{g}_e^*(t_k)g_e^*(t_k)^{-1}$$

3. If we are *not* given a velocity trajectory, we can use the Lie group logarithm to obtain the desired twist trajectory:

$$\xi^b(t_k) = \ln_{\Delta t}\left(g_e^*(t_k)^{-1}g_e^*(t_{k+1})\right)$$
$$\text{or}$$
$$\xi^s(t_k) = \text{Ad}_{g_e(t_k)}\xi^b(t_k)$$

   We can compute this logarithm using the matrix logarithm with python/Matlab functions as:

$$\ln_{\Delta t}(g) = \text{logm}(g) \cdot \frac{1}{\Delta t}$$

4. Next, we translate the desired task-space waypoints to the associated desired joint-space waypoints using inverse kinematics. The preferred method for this is to use resolved-rate inverse kinematics since it is numerically stable in situations where there could be multiple inverse kinematics solutions. The basic idea of resolved rate is to use a first-order accurate integration technique to update the joint angles:

$$\theta(t_{k+1}) = \theta(t_k) + \dot{\theta}(t_k)\Delta t$$

   with $\dot{\theta}(t_k)$ obtained using the inverse of the manipulator Jacobian:

$$\dot{\theta}(t_k) = (J^b(\theta(t_k)))^\dagger \xi^b(t_k)$$
$$\text{or}$$
$$= (J^s(\theta(t_k)))^\dagger \xi^s(t_k)$$

Time-Scaling Functions

If we want to enforce straight-line paths, we can track these paths with a time-scaling function to still enforce smooth velocities and accelerations. The most common time-scaling function is the cubic polynomial:

$$s(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3$$

with

$$\begin{aligned} a_0 &= 0 \\ a_1 &= 0 \\ a_2 &= 3/T^2 \\ a_3 &= -2/T^3 \end{aligned}$$

Notice that these coefficients are assuming that we start and end at rest.

Then, using this time-scaling, we can obtain straight-line paths in the joint-space using:

$$\theta(s) = \theta_i + s(\theta_f - \theta_i), \quad s \in [0, 1]$$

Alternatively, for straight-line paths in the task-space, we can use:

$$\begin{aligned} g(s) &= g(0)\exp(\xi s) \\ &= g_i^* \exp(\ln(g_i^* g_f^*)s) \end{aligned}$$

which can be broken down into:

$$\begin{aligned} p(s) &= p_i + s(p_f - p_i) \\ R(s) &= R_i \exp(\ln(R_i^T R_f)s) \end{aligned}$$

Wrenches and Forces

The manipulator Jacobian is also used to map joint torques to end-effector wrenches. The manipulator Jacobian is used to map joint torques to end-effector wrenches:

$$\begin{aligned} \tau &= J_b^\top(\theta) F_b \\ &\text{or} \\ &= J_s^\top(\theta) F_s \end{aligned}$$

Here, we define a wrench to be the combination of the end-effector linear forces and rotational moments:

$$F = \begin{bmatrix} f \\ \tau \end{bmatrix}$$

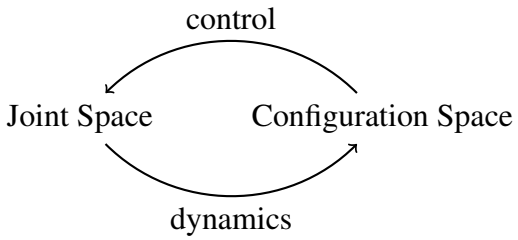
We can compute *equivalent* wrenches applied to different points on the manipulator by changing the frame of reference using the adjoint:

$$F_b = \text{Ad}_{g_{ab}}^\top F_a$$

with F_a being a wrench applied at the origin of coordinate frame a and F_b being the equivalent wrench applied to the origin of the coordinate frame b.

Dynamics and Control

The second derivative intepretation of our mapping between joint space and configuration space is:



with the explicit mappings:

$$\begin{aligned} \textit{forwards dynamics:} \quad & \ddot{\theta} = M^{-1}(\theta)(\tau - h(\theta, \dot{\theta})) \\ \textit{inverse dynamics:} \quad & \tau = M(\theta)\ddot{\theta} + h(\theta, \dot{\theta}) \end{aligned}$$

Here, these equations use the robotics equations of motion:

$$\tau = M(\theta)\ddot{\theta} + \underbrace{C(\theta, \dot{\theta})\dot{\theta} + G(\theta)}_{h(\theta,\dot{\theta})}$$

where $\theta = \mathbb{R}^n$ is the vector of joint angles, $\tau = \mathbb{R}^n$ is the vector of joint torques (and forces if linear actuators), $M(\theta) = \mathbb{R}^{n \times n}$ is a symmetric positive-definite mass matrix, $C(\theta, \dot{\theta}) = \mathbb{R}^{n \times n}$ is the Coriolis matrix, and $G(\theta) = \mathbb{R}^n$ is the gravity vector. As shown, the Coriolis matrix and the gravity vector are often lumped together into a generalized vector h. As stated in LP, these expressions can be "extraordinarily complex" despite their simple form.

These equations can be used to design controllers for our manipulator. The most common controllers are feedback controllers and feedforward controllers. The pseudocode for feedforward control is:

```
time = 0                                                  // dt =  cycle time
loop
    [qd, qdotd, qdotdotd] = trajectory(time)              // trajectory gen.
    tau = M(qd)*qdotdotd + C(qd, qdotd)*qdotd + G(qd) // calculate dynamics
    commandTorque(tau)
    time = time + dt
end loop
```

The pseudocode for feedback control is:

```
time = 0                                                    // dt =  cycle time
eint = 0                                                    // integral error
qprev = readJointAngles()                                   // init. joint angle q
loop
    [qd, qdotd] = trajectory(time)                          // trajectory gen.
    q = readJointAngles()                                   // read joint angle q
    qdot = (q-qprev)/dt                                     // calculate joint vel
    qprev = q

    e = qd - q                                              // position error
    edot = qdotd - qdot                                     // velocity error
    eint = eint + e*dt                                      // integral error

    tau = Kp*e + Ki*eint + Kd*edot                          // PID control
    commandTorque(tau)

    time = time + dt
end loop
```

Equations Page

Provided below are the equations which you will be given on the Final Exam.

Rotation Matrices

In SO(3), the rotation matrices corresponding to rotations about the x, y, and z axes are:

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix}, \quad R_y(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix},$$

$$R_z(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

In $SO(2)$, a rotation matrix has the form:

$$R(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

Matrix Adjoint

$$\text{Ad}_g = \begin{bmatrix} R & [d]_\times R \\ 0 & R \end{bmatrix} \quad \text{assuming } \xi := \begin{bmatrix} v \\ \omega \end{bmatrix} \qquad \text{(Convention in MLS)}$$

or

$$= \begin{bmatrix} R & 0 \\ [d]_\times R & R \end{bmatrix} \quad \text{assuming } \xi := \begin{bmatrix} \omega \\ v \end{bmatrix} \qquad \text{(Convention in LP)}$$

Rodrigues' Formula

$$\exp([\omega]_\times \tau) = \begin{cases} I + \frac{[\omega]_\times}{\|\omega\|}\sin(\|\omega\|\tau) + \frac{[\omega]_\times^2}{\|\omega\|^2}(1 - \cos(\|\omega\|\tau)) & \text{if } \omega \neq 0 \\ I & \text{if } \omega = 0. \end{cases}$$

Matrix Exponentials

The full exponential map for $SE(3)$ is:

$$\exp(\hat{\xi}\theta) = \begin{bmatrix} e^{[\omega]_\times \theta} & (I - e^{\hat{\omega}\theta})[\omega]_\times \hat{v} + \omega\omega^T v\theta \\ 0 & 1 \end{bmatrix}$$

For revolute joints in $SE(3)$, the exponential map simplifies to:

$$e^{\hat{\xi}\theta} = \begin{bmatrix} e^{[\omega]_\times \theta} & (I - e^{[\omega]_\times \theta})q \\ 0 & 1 \end{bmatrix}$$

For prismatic joints in $SE(3)$, the exponential map simplifies to:

$$\exp(\hat{\xi}\theta) = \begin{bmatrix} I & v\theta \\ 0 & 1 \end{bmatrix}$$

Law of Cosines

Given a triangle with sides a, b, and c, the law of cosines provides us with the length of side c given the angle opposite of side c (angle C):

$$c = \sqrt{a^2 + b^2 - 2ab\cos(C)}.$$

Another way of stating this is:

$$C = \cos^{-1}\left(\frac{a^2 + b^2 - c^2}{2ab}\right).$$

Manipulator Jacobian

The spatial manipulator Jacobian is:

$$\begin{aligned} J_s &= \begin{bmatrix} \xi_1 & \text{Ad}_{e^{\hat{\xi}_1\theta_1}}\xi_2 & \cdots & \text{Ad}_{e^{\hat{\xi}_1\theta_1}\cdots e^{\hat{\xi}_{m-1}\theta_{m-1}}}\xi_m \end{bmatrix} \\ &= \begin{bmatrix} \frac{\partial g_1}{\partial\theta_1}g_1^{-1} & \text{Ad}_{g_1}\frac{\partial g_2}{\partial\theta_2}g_2^{-1} & \cdots & \text{Ad}_{g_1 g_2 \cdots g_{m-1}}\frac{\partial g_m}{\partial\theta_m}g_m^{-1} \end{bmatrix} \end{aligned}$$

The body manipulator Jacobian is:

$$\begin{aligned} J_b &= \begin{bmatrix} \text{Ad}^{-1}_{e^{\hat{\xi}_1\theta_1}\cdots e^{\hat{\xi}_m\theta_m}g_0}\xi_1 & \cdots & \text{Ad}^{-1}_{e^{\hat{\xi}_m\theta_m}g_0}\xi_m \end{bmatrix} \\ &= \begin{bmatrix} \text{Ad}^{-1}_{g_2\cdots g_{m+1}}g_1^{-1}\frac{\partial g_1}{\partial\theta_1} & \cdots & \text{Ad}^{-1}_{g_{m+1}}g_m^{-1}\frac{\partial g_m}{\partial\theta_m} \end{bmatrix} \end{aligned}$$

Skew-Symmetric Matrices

The skew-symmetric matrix of ω is:

$$\hat{\omega} = \begin{bmatrix} 0 & -\omega^3 & \omega^2 \\ \omega^3 & 0 & -\omega^1 \\ -\omega^2 & \omega^1 & 0 \end{bmatrix}.$$

TEST TEST TEST

$$J(\theta) = \begin{bmatrix} \cos(\theta_1) & 0 & 1 \\ -\sin(\theta_1) & \theta_2 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

$$J(\theta) = \begin{bmatrix} & \cos(\theta_1) & 1 \\ \sin(\theta_1) - \sin(\theta_1 + \theta_2) & -\sin(\theta_1) & 0 \\ & 1 & 0 \end{bmatrix}$$

$$\hat{\xi} = \begin{bmatrix} 0 & 0 & 0 & -1 \\ 0 & 0 & 2 & 0 \\ 0 & -2 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$